

Quasi-Decentralized Cyber-Physical Fabrication Systems—A Practical Overview



Ilija Vukorep and Anatolii Kotov

Abstract Building an effective cyber-physical system is difficult due to the overall complexity of technologies on their own, challenges with the interaction between all parts of workflow and applicability issues. This makes the real-world application of complex cyber-physical systems only available to the big industry parties or high-tech startups, leaving small and medium-sized businesses behind. Therefore, the democratization of such applications is a reasonable goal to achieve. Cyber-physical systems of this kind are in the experimental stages and incorporate robotics, IoT, materials science, visual and 3D-scanning techniques, and machine learning (ML) tools all under the heading of Industry 4.0. This paper covers specific practical approaches in several application fields using robotics and IoT. We use custom-built hardware and software setups together with standard frameworks and the MQTT protocol for different applications. Due to the practice-driven approach, the paper will illustrate both positive and negative effects. We describe the use of our systems in several case studies: Multi-layer automated robotic concrete spraying using ML, IoT spatial awareness via sensors (such as Lidar, Kinect), robotic multi-axis milling and quasi-autonomous robot movements. A unifying issue is a decentralized approach of modular IoT elements that we grouped to achieve specific tasks. The paper illustrates how these elements exchange data and communicate, and all services are controlled on each computer instances, connected to an IoT network to ensure a high level of system stability.

Keywords Cyber-physical system · Digital fabrication · Industry 4.0 · Decentralized digital services

United Nations' Sustainable Development Goals 9. Industry Innovation and Infrastructure · 12. Responsible Consumption and Production

I. Vukorep (✉) · A. Kotov
Chair of Digital Design Department, Brandenburg University of Technology, 03046 Cottbus,
Germany
e-mail: ilija.vukorep@b-tu.de

1 Introduction

Our paper discusses practical approaches to robotics and Internet of Things (IoT) in several application fields. In order to increase the reusability of such cyber-physical setups, a combination of custom hardware and software, as well as standard frameworks and protocols are used for different applications. We implement principles of Industry 4.0 paradigm by using several architecture defining principles such as microservices, applied robotics and ML. There are several examples of similar setups of using industry 4.0 principles in robotic fabrication [1, 2]. Micro-services allow us to isolate key services and machines, enabling us to create decentralized, modular IoT elements that can be grouped together to achieve specific tasks. Furthermore, microservices are used within the software and hardware domains, allowing for better communication and data exchange between different segments. This further enhances the integration of our cyber-physical systems, allowing us to achieve greater levels of automation and efficiency. This concept embodies one of industry 4.0's main strengths, which is the democratization of complex tools.

We are discussing some typical services in detail used in our applications and providing links to some own developed libraries and components on GitLab [3]. Our system is demonstrated in several case studies: Multi-layer automated robotic concrete spraying using machine Learning (ML), IoT, spatial awareness using sensors (such as LiDAR, Kinect), robotic multi-axis milling, and in teaching scenarios. A common theme is a decentralized approach of modular IoT elements that we group to achieve specific tasks. One of the tasks is the concrete spraying case study that is aiming to reduce the weight of produced elements and increase the potential design space through its capability to produce free-form shaped parts. Its production incorporates a wide range of machine control, space observation and data processing segments. The paper will describe how these segments exchange data and communicate with each other. In the quasi-autonomous case study section, we will cover the interaction of a pair of robots, where one robot is performing target action, while the second is following the first one via dynamic observation. Micro-services are used both in a software and hardware sense—both key services and machines are isolated. We start by defining the problem statement, which discusses the use of Robot Operating System (ROS), a leading system for developing cyber-physical systems. The following chapter is about categorizing various types of cyber-physical systems in terms of their internal organization and communication of processes.

2 Problem Statement

There are several systems that can help in developing automatized and robotic applications. One of the most popular and robust is ROS—robot operating system, a framework for various robots founded 2007 at Stanford University. It is suitable for heterogeneous clusters with a highly developed messaging system with low

latency. Its publish-subscription communication management is even near real-time in the new ROS 2 version. Furthermore, its services run decentralized, so demanding processes can execute remotely. ROS' lively community is providing libraries for all kinds of hardware. Communication with external non-ROS clients is possible (Websockets, mqtt). The ROS-industrial is an extension of ROS and provides open source industrial general and vendor specific libraries in an industrial context. The drawback of such a powerful system is that the implementation can be very complex. In a typical academic scenario, where experiments are performed with light and atypical configurations, setting up ROS with its enormous overhead can be extremely time-consuming. Although ROS covers a big range of hardware and allows implementation of one's own libraries, installing unsupported sensors and actuators inside of ROS proved to be unrealistic in our context of projects.

The setup of a construction-related robotic system requires the creation of several services that can work robustly and independently and have a clear way of interacting. As most robotic setups are different, those services should operate in different constellations. The software can be written in different languages, and the services can run on completely different hardware. With this, to distribute the development of single parts to a wider range of people is easily possible, in our case to students and academic researchers. When working with parametric geometry, such as connecting Rhino 3D Grasshopper results to a robotic setup with many sensory inputs and complex actuator control, this flexibility is especially important. Another important feature is also the ability to work remotely, via VPN, when needed, with visual feedback.

3 State-of-the-Art

In the architectural and construction context, most academic experimental robotic setups are very narrowly and pragmatically tailored to solve some specific scientific problem. The majority of related research papers do not detail their technical setup, but a few of them mention their software equipment and this has been analyzed. Generally, we can divide architecture- and construction-oriented robotic setups into several groups:

1. **ROS as a central unit.** ROS is often used for its integrated and decentralized service platform and communication protocol. In [4] research, a collaborative human-robot construction system is developed around a ROS computational core and its ability to communicate with virtual reality in Unity, sensor data, and the robot. In the work of [5], ROS is exchanging data with MATLAB Robotics System Toolbox for path creation. In [6] ROS is used together with MoveIt for trajectory planning of the robots. In [7] Choreo is used along with ROS-industrial and Move-it to plan motion and choreography.
2. **ROS as one of several parallel services.** The work [8] involves modeling in Rhino, kinematic modeling in the COMPAS FAB package, and ROS as a robot controller with communication via Roslibpy. This library was developed

at the ETH Gramazio. Kohler and is based on WebSockets and bridges between services. For [9] GH/Lunchbox ML services, as well as serialized data in JSON format, are implemented.

3. **Decentralized system with heterogeneous communication.** The work [10] uses individual services for scanning (python), ML analysis (Tensorflow, python), modelling (GH/Rhino) and path planning (RoboDK) without a specific communication protocol. In this study, [11] Rhino for modelling, GH to simulate robotic processes, Unity for VR-visualization, DynamoDB at Amazon Web Services (AWS) for data handling and data storage are used and heterogeneously connected.
4. **Decentralized system with specific server communication.** Described in [12] is a robot setup controlled by a self-written python server (XML, TCP/IP Ethernet) that transmits data between services (clients: Python algorithms, Rhino visualization, camera, and robot). In [13] the communication is arranged through Java in Processing and an UDP protocol exchanging data with the Scorpion plugin in Rhino/GH for path creation and transmission to the robot.

These categories each have their own advantages and impact on scalability, modularity, and reusability. A decentralized system with a Message Queuing Telemetry Transport (MQTT) protocol is described in this article based on experiences gained in a variety of scenarios. The protocol serves as a basis for IoT development, as all clients and services in an IoT system communicate via the web. These services will be described in more detail within the following chapters.

4 Components/Services

Our system uses a modular microservices architecture. Most commonly, this term describes the organization of software complexes and information systems. Contrary to a monolithic approach where all the code is merged together, the microservices concept divides a program into several independent components (services) that can run on multiple/divided platforms and have a unified communication vocabulary. If one subsystem fails, the whole system will be less likely to crash, what improves stability. Another advantage is a possible update/change of individual components inside the system without having to rebuild the entire system. There are currently many distributed online systems with millions of users using it. As a basis for further development, this approach was chosen due to the need for stability, modularity (reusability), and scalability.

4.1 MQTT Broker—Infrastructure Server

The core element of this decentralized cyber-physical system is the IoT data exchange protocol (MQTT). In contrast to *XML* over *TCP/IP*, *Websockets*, *UDP* or other previously mentioned communication protocols, is that it can deliver messages with requested quality of service (QoS). This includes fire and forget, at least once and exactly once. QoS can help when connections are unstable or we have critical command execution procedures (as running a robot). It's supplying sufficient speed and it's lightweight. The principle of subscription and publishing over topics is similar to ROS' handling messages. The issues of security around MQTT are not explicitly handled in this article and need special care by using client identification rules. The broker runs in the background inside a network. In our settings, we ran a *Mosquitto* broker (mosquitto.org) on *Raspberry Pi 4* that automatically activates the service on booting. It can run on any OS system with a known IP address to the clients. For testing purposes, we provide a self-developed MQTT server on local-host running inside *GH/Rhino* as a component. Furthermore, we wrote a MQTT subscription and publication component for *GH/Rhino* [3].

4.2 Dashboard

To manage all processes, we could theoretically use any MQTT-browser like *MQTT-lens* [14] but this will not give us a good overall overview of incoming and outgoing information. For visualization and administration of all processes, a central dashboard is necessary. Fig. 1 is showing an example of such a dashboard with several pages. The best method of a dashboard that is compatible for diverse use cases is to keep it growing as services are being added. This means that services that are not used are automatically disabled in some applications. This can be done by internal checking if services are available (online).

Key features of a dashboard are:

- Robot control, with some predefined robot positions or movements, manually loading robot scripts or others,
- System observation, key vital data of robot TCP position, availability of hardware and program components,
- Visual observation from connected cameras,
- Control of additional machines as scanners, grippers,
- Database connection with data presentation and editing options.

Our dashboard is built with *Python* and *Flask* backend programming framework that can easily be scaled as it uses templates and sophisticated data interaction. These packages can run on *Raspberry Pi 4* or any other computer (also together with the broker) in the network and can be reached at the local network.

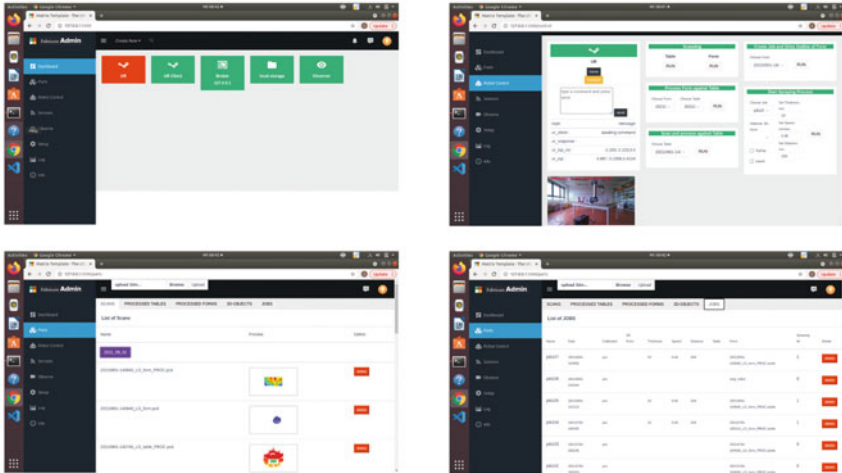


Fig. 1 Dashboard pages: status, robot control, scan data view and info page

4.3 Robot—IoT Interface

One central part of the IoT robotic setup is the connection of the robot to the rest of the system. For this connection, we use an interface service that is transferring incoming data directly to the robot over an Ethernet/Lan connection and vice versa, publishing information like robots joint and TCP positions data. As this interface is embedded in a wireless network to other clients, huge flexibility of robotic control is provided, which means we can control the robot without being directly connected to it.

As every robotic company has got its own I/O solution, this interface has to be tailored individually to robotic families. We tested this setup with *Universal Robots* running the interface on a Raspberry Pi 4 and written in Python. All outgoing information about the robot is read by the IO Modbus protocol of the UR robots and broadcasted over MQTT.

Additional machines connected to the robot are also accessible through this interface as part of the script that is transferred to the robot. This transfer incorporates also some extra features such as comparing the robot position with the target position or relative and absolute movements and some special positions, like home, self-test or tool changing position.

4.4 Data Management

Data management is not organized as a single service but can be found in all other services and processes. In our cyber-physical arrangements, we have four data streams that handles: Configuration files of the setup, incoming (i.e. sensor) data, processed data and logging data. Configuration files are actual settings (home position, special robotic positions, broker addresses, storage credential data) stored as JSON-files. Besides this, we also have environment and tool data stored as Rhino 3 dm-files. Both types are inside the folder structure of the *dashboard*.

Incoming data are files made by sensors and are mostly of high volume. Those are saved in a project folder with a time-stamp and additionally automatically processed or triggered by incoming commands. Those processed files are then saved in additional folders like thumb views of scanning or images, automatically generated scripts or other project specific data. All processed data that are part of closed procedure, are saved in the job-folder. This decentralized database-free approach is working well in closed networks but there are security issues when we expose them in open networks. For this usually some professional data handling services can be used like AWS S3 services that is also handling user-right-management and some data processing through APIs. In our setup we used a *Resilio Sync—BitTorrent* protocol for data synchronization as this proved to be fast.

Logging is an important step toward error tracking and statistical analysis. Some of the services in our cyber-physical system are logging their processes. Advisable would be a central logging device, even organized as an own service, that is collecting logging data from other services with the capability of analytic display in dashboard or other viewer.

4.5 Vision

Cameras can be used for tracking movement, security control, position diagnosis or simple documentation purposes. The application will define if camera stream will be recorded or not, if the processed image will be directly processed or high computation is necessary (i.e. OpenCV postprocessing). Any stream data client can be connected to the network and then retrieved from the dashboard. Based on streamed data it's possible to detect movement, risk situations (and use MQTT for signaling) or incorporate customized recognition of elements. In our case we have used the camera for finding edges of the scanning area. For simple low resolution processing, a camera can be connected to a Raspberry Pi 4, which can run OpenCV4, and transfer the data over Wi-Fi. For higher resolution data streams an Ethernet connection is necessary. Remote control of the system is also facilitated by vision.

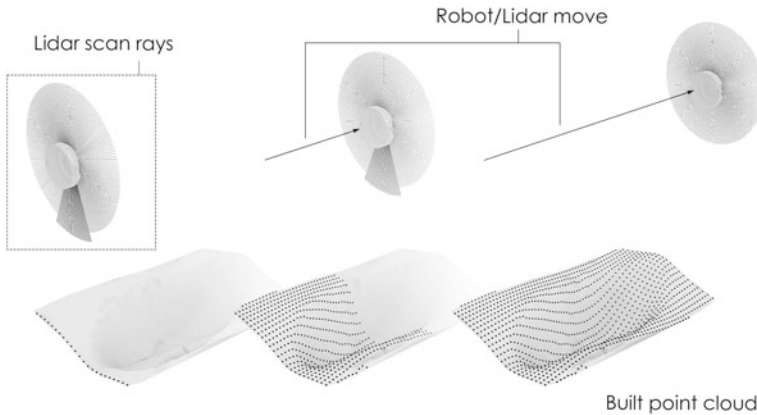


Fig. 2 Illustration of the scanner movement over the observed object

4.6 Sensing

The complexity of the task and models meant that we had to use different scanning systems in our project. We did tests with *Kinect*, *azure Kinect DK* and *RPLidar* scanners. All three have different capacities and options of sensing the environment.

4.6.1 Rotation Lidar Scanner

Rotation lidar scanner like is measuring distance to an object at some specified angle. As it rotates, it can only measure in slices, meaning if we want to measure distances in all 3 axes, the lidar scanner has to move. In combination with a precise movement of a robot we can have a very effective scanner that can, in some cases, make better scans of areas that are obscured, deep objects or with cantilevers (Fig. 2). We connected our RPLidar A2M8 scanner to a Raspberry Pi 4. Together with the UR-robot, both were triggered by an MQTT command [3]. Scanning results grouped with the coordinates of the scanner positions were saved in the incoming data folder and automatically further processed. This data processing usually depends on the application and in our case it consisted of deleting the unnecessary data and transforming the rest into a 3D point cloud.

4.6.2 Azure Kinect SDK Scanner

For certain tasks, such as real-time scanning for object presence and control, a moving lidar is not the best solution. Therefore, a depth camera coupled with a RGB-camera seemed like a good option [3]. We chose an *Azure Kinect SDK* scanner that is connected to a computer with the *Ubuntu OS*. Even though the hardware

and software requirements are high, the scanning process worked well because we needed a high level of precision without a high resolution. As the RGB-stream can be read separately, we use the camera also for documentation purposes. Several MQTT commands are running the scanner: RGB-single shot, RGB-stream, hi-resolution scan, low-resolution scan. The results are stored in the incoming data folder and are automatically processed like creating thumb views of the point clouds or detecting objects.

In our case we mounted the scanner above the active zone of the robot, see Fig. 5. In the usage examples described later, the scanner serves for sensing the actual state of robot production. For this, every process step is scanned and the object scan are compared after the process step is finished. Before the process starts, a calibration scan is executed that marks the clean table and initializes the process.

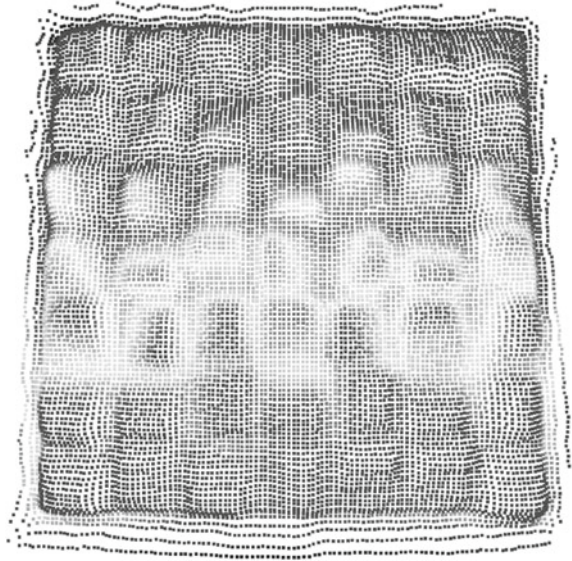
4.7 Point Cloud Processing Service

When scan data is stored in the incoming folder, several application dependent processing steps are necessary. Usually object have to be detected from out of a messy point cloud and for this the following functions are executed: (1) the empty table scan detects a plane, (2) the object scan is been cleaned against this plane, (3) the result is clustered so some flying points eliminated, (4) the outline of the object is found, that all further scans can use for the eliminating unnecessary point data. Although this service depends on the application we wanted to develop, it as a module that can be reused in other applications. The algorithms are triggered by MQTT-commands from the dashboard. Most processing work is done in the background while the dashboard lists the objects in the browser views. Fig. 3 shows a 3D scanned mould after the point cloud cleaning procedures.

4.8 Grasshopper Components

Since the wide adoption and the abilities of Rhino/Grasshopper, it was essential to create viable MQTT components for Grasshopper. During different development phases we also used GH/Python connected modules, but that was not offering enough scalability, since one has to install and configure a Python environment for any new machine. GH components are simpler to use and also better in terms of response of the GUI since they're natively compiled from C# and Grasshopper is built on.NET technology. The components for Rhino 6 and 7 can be downloaded from GitLab [3].

Fig. 3 Mold 3d scan example automatically subtracted from the surrounding environment



4.9 Path Planning Service

The goal of the path planning service is to provide a viable adaptive trajectory over a given surface keeping in mind execution of additional tasks. Those tasks are the execution of actions or activating hardware connected to the robot depending on path and/or surface properties. The path planning are then combined with action augmentation. This means, in certain positions or areas robot activates connected hardware (e.g. use gripper or run the concrete pump).

A naive path planning approach can be defined as a *zig-zag pattern* based on a given shape (Fig. 4). However, such an approach is ignoring the curvature of the target surface. In this case it is not adaptive, and this can cause problems with complex shapes and where actions require precision. We calculate simple patterns based on the previously mentioned outline form without any usage of additional 3D programs.

An alternative approach would be the use of ML methods with the aim to be adaptive to any form and be compatible with imperfections of the scanned object. We use a *Multi-Objective Optimization (MOO)* strategy in conjunction with other ML methods. These methods are proven to be useful as one of the tools in architectural production [14]. However, to apply those methods, the transformation of several stages of data is necessary. The path planning between different zones can be also formulated with the *Travel Salesman Problem (TCP)*, which enables us to find the optimal path. To solve it, we're using internal Grasshopper components with custom Python processing. Thus, we're solving the problem of uniform distribution of the points over the given surface with desired in-between distance (a derivative of action radius and distance between TCP and surface). As a result, a waypoint graph with equal edge length based on the surface of any curvature is generated. Using the

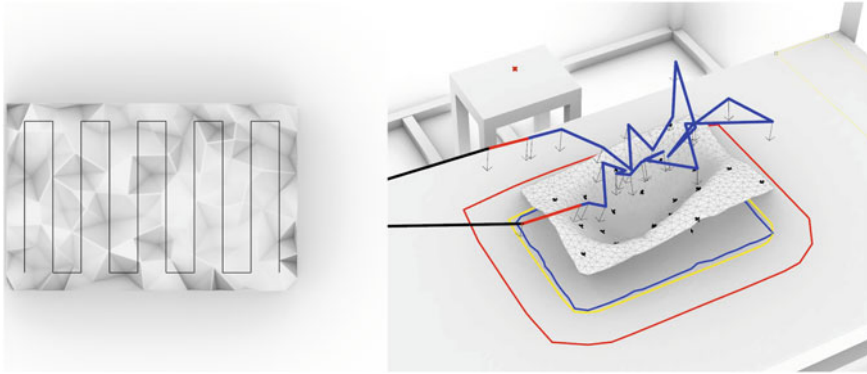


Fig. 4 Left: naive zigzag approach, right: ML approach with the determination of the working area for the path planning algorithm as well as for dynamic hardware control settings (e.g., we stop air & concrete pumps while the robot is travelling outside/between red and blue line, and enable it over islands/inside blue line)

previously generated graph and applying TCP guided local search algorithm, we’re able to find the optimal path. In our examples we are using different script generation software like RoboDK (roboDK.com) and Robots (github.com/visose/Robots) running in Rhino/GH instances. Those are activated by MQTT commands that sends the number of the job so the identification of the files in the job folder is possible.

To ensure that the robot movement is safe, the path is being simulated in a duplicated virtual environment where the virtual robot and all the robotic movements are synchronously simulated prior to the machine execution. Thus, we’re testing that the given path can be executed, that all points can be reached, that there are no singularities and collisions (and signaling an error over MQTT). As a final part of the program, the resulting script is being augmented according to the target setting for acceleration and speed of the robot’s linear and joint movements’ settings.

4.9.1 Digital Twin

During the implementation of certain use strategies we found it very useful to implement a digital twin of the URs that we use in the lab. This can help us to prevent singularities or collisions with itself, other robots or the environment. The difference between a digital twin and a simulation is that it simulates the setup in real time and with all possible MQTT inputs that is streamed over the *Robot—IoT interface*. Each simulation is using individual UR calibration files, so we can be sure that it’s 1:1 to the real setup. The other benefit of using digital twins is the fact that one can develop programs for robots without the physical availability of the systems (e.g. develop robot control system not from the robot lab, but from home office). Later, the algorithms and models tested on digital twins can be propagated onto real hardware. For implementation we are using Python for MQTT-side communication and

control, sharing part of the codebase with a *Robot—IoT interface* and RoboDK, all running on a windows machine inside a Rhino/GH instance. The limitations lie in limited sensor (vision and scanning) data or the simulated production process that can be complex.

5 Applications

5.1 Adaptive Robotic Concrete Spraying

Concrete spraying is one of the promising technologies that aim to reduce the weight of produced elements and increase the possible design space through its feasibility in producing free-form shaped parts. Even though thin-formed concrete shells production is efficient for many situations, a variable thickness would increase applications. First situation is resulting from the inconsistent process of spraying. It often suffers from an interruption in concrete feeding that the pump is pushing through the tube. As these interruptions are happening during the robot movement, the applied material unintentionally varies in thickness. The second situation comes from the structural properties of the element that sometimes demands to have thicker or slimmer areas depending on its structural needs.

Using ML methods this system offers different spraying strategies: Using target thickness or matching the target form. The spraying is done automatically using 3d scanning and adaptive robot path planning (Fig. 5). The software subsystem consists of several parts:

- Point-cloud scanning of mould and scanning the processing phases,
- Form comparison/matching—comparison of the given form to target goals,
- Robot path planning component—optimal robot spraying trajectory,
- Robot—IoT interface for execution of desired spraying path, controlling pump and air valve.

While there may be simple forms, the high curvature forms' surfaces present a challenge. This requires a complex path plan strategy as the mechanical movements of the robot should not collide with the mold during the spraying process as well as to follow the goal parameters of the target concrete shape. The outcome of the research can be found [15].

5.2 6-Axis Milling

Traditional CNC-milling is usually limited to milling in 3 axes. However, attaching a milling machine to a 6 axis robot arm allows to achieve much more versatile milling.

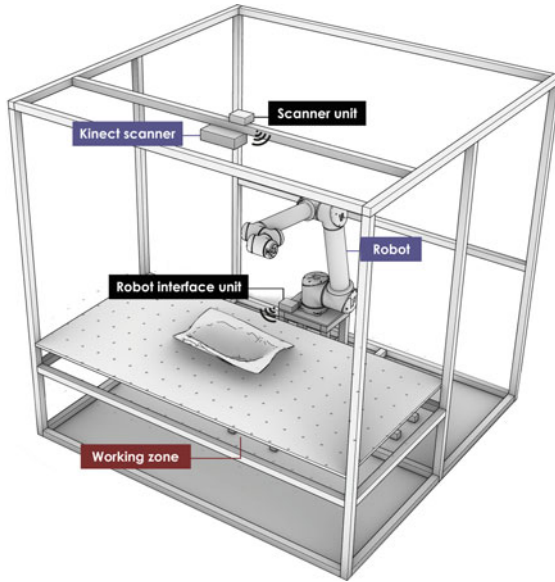


Fig. 5 Hardware and software elements of a cyber-physical concrete spraying system

One of the key abilities is so-called “undercutting”, the movement when a robot drill goes under an already existing form without destroying it (Fig. 6).

In our setup the robot was controlled via a Raspberry Pi with our software. The most challenging part of this application was the software that forms the movement trajectories in 6 axis movements, to check if the form is feasible to milling and the robot arm and milling drill will not collide with the milled model itself. Furthermore, there are also issues of visibility of milling paths on the target surface in case of sophisticated target form. Despite the proof-of-work state, developing a full-scale application seems challenging due to the complexity of robot path planning and challenges with fine milling of complex curvatures.

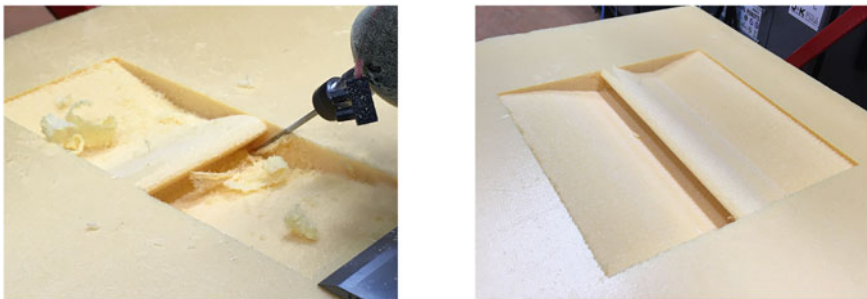


Fig. 6 Left: undercutting maneuver, right: resulting mold

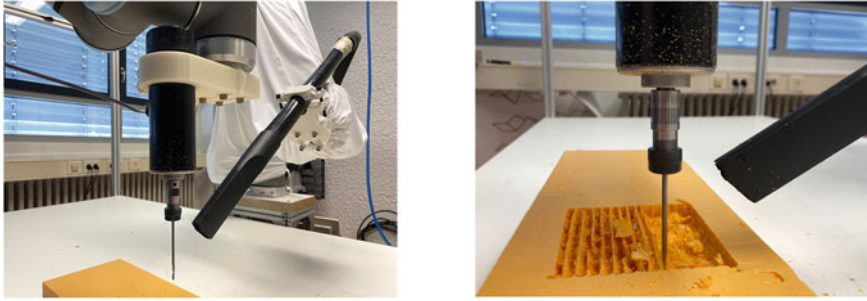


Fig. 7 Left: system setup, right: milling-following-vacuuuming

5.3 Milling with Real-Time Relative Robot Movements

Milling produces a lot of dust, which is a problem. In this instance, we are solving this with a robotic IoT toolset. Next to a milling instrument attached to one UR, we added another actor; a new UR with the vacuum nozzle that is designed to follow the milling TCP of the first UR robot to vacuum all dust generated during the milling process.

Hardware setup consists of UR robots each connected with Raspberries. During the implementation of this solution we found the need to create the 1:1 digital twin of the acting robots in the environment due to the fact that it was more convenient to debug controlling algorithms in a digital way. UR1 is milling, while UR2 is following UR1 TCP via constant reading real-time position and axis data (Fig. 7). Using this data UR2 is correcting its position.

5.4 Teaching

Teaching digital design methods in universities often involves the use of robots. The robots are usually housed in cages and separated from the workshop area to minimize injury risks. Several groups of students can work with the robot and develop experimental setups. We found that the possibility to wireless control the robot over MQTT can help in the workflow as there is no need to always re-switch the cable to the robot control unit. Still, as everybody can control the robot from their computer and if the right MQTT topic name and the broker's address is known, this can increase the risk of uncontrolled sending commands to the robot. This type of dangerous interference can be reduced with strict rules in robotic workshops (time-slots for groups) and vision-based recognition to signal, when the robot area is free of people.

6 Results and Future Work

In this paper we described a decentralized cyber-physical setup based on services that can be easily reused for various applications. As the main communication protocol is MQTT with good remote capabilities it can serve as a classical IoT example. Due to security and latency issues, the described setup has still no direct industrial potential but as the system is modular, elements can be evolved towards professional use cases. It is perfect for academic workshops where a lot of task-driven pragmatic and quick development is possible.

All presented services can be improved, especially where real-time is necessary (i.e. controlling safety measures in high risk environments). We noticed 300–500 ms latency when controlling the robot with a gamepad or in the milling-following-vacuuming application example. Also the data management can incorporate industry-grade data handling services such as AWS S3 services, Azure or Google Cloud to avoid exposure of data with a safe user-management. Logging can be incorporated as a single central service together with a good analytic visualization. This setup was tested only with the UR robot family. The interaction with other robots should be further developed and tested. Some of the libraries and GH components are publicly available at GitLab [3].

References

1. Betti, G., Aziz, S., Rossi, A., Tessmann, O.: Communication landscapes. In: *Robotic Fabrication in Architecture, Art and Design*, Springer International Publishing, Cham, pp. 74–84 (2019). https://doi.org/10.1007/978-3-319-92294-2_6
2. Reinhardt, D., Haeusler, M.H., London, K., Loke, L., Feng, Y., De Oliveira Barata, E., Firth, C., Dunn, K., Khean, N., Fabbri, A., Wozniak-O'Connor, D., Masuda, R.: CoBuilt 4.0: Investigating the potential of collaborative robotics for subject matter experts. *Int. J. Archit. Comput.* **18**, 353–370 (2020). <https://doi.org/10.1177/1478077120948742>
3. https://gitlab.com/Digitales_Entwerfen/pub_repository
4. Wang, X., Liang, C.-J., Menassa, C., Kamat, V.: Real-time process-level digital twin for collaborative human-robot construction work. In: Presented at the 37th International Symposium on Automation and Robotics in Construction, Kitakyushu, Japan (2020). <https://doi.org/10.22260/ISARC2020/0212>
5. Gill, R., Kulić, D., Nielsen, C.: Path following for mobile manipulators. In: Bicchi, A., Burgard, W. (eds.) *Robotics Research: Springer Proceedings in Advanced Robotics*, vol. 2, pp. 527–544. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-60916-4_30
6. Kaiser, B., Littfinski, D., Verl, A.: Automatic generation of digital twin models for simulation of reconfigurable robotic fabrication systems for timber prefabrication. In: Presented at the 38th International Symposium on Automation and Robotics in Construction, Dubai, UAE (2021). <https://doi.org/10.22260/ISARC2021/0097>
7. Huang, Y., Carstensen, J., Tessmer, L., Mueller, C.: Robotic extrusion of architectural structures with nonstandard topology. In: *Robotic Fabrication in Architecture, Art and Design 2018*, pp. 377–389. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-319-92294-2_29

8. Ercan Jenny, S., Lloret, E., Gramazio, F., Kohler, M.: Crafting plaster through continuous mobile robotic fabrication on-site. *Constr. Robot.* **4**, 1–11 (2020). <https://doi.org/10.1007/s41693-020-00043-8>
9. Ercan Jenny, S., Lloret-Fritschi, E., Jenny, D., Sounigo, E., Tsai, P.-H., Gramazio, F., Kohler, M.: Robotic plaster spraying: crafting surfaces with adaptive thin-layer printing. *3D Print. Addit. Manuf.* 3dp.2020.0355 (2021). <https://doi.org/10.1089/3dp.2020.0355>
10. Nicholas, P., Rossi, G., Williams, E., Bennett, M., Schork, T.: Integrating real-time multi-resolution scanning and machine learning for conformal robotic 3D printing in architecture. *Int. J. Archit. Comput.* **18**, 371–384 (2020). <https://doi.org/10.1177/1478077120948203>
11. Ravi, K.S.D., Ng, M.S., Ibáñez, J.M., Hall, D.M.: Real-time digital twin of on-site robotic construction processes in mixed reality 8. (2021)
12. Dörfler, K., Rist, F., Rust, R.: Interlacing. pp. 82–91 (2013). https://doi.org/10.1007/978-3-7091-1465-0_7
13. Elashry, K., Glynn, R.: An approach to automated construction using adaptive programming. pp. 51–66 (2014). https://doi.org/10.1007/978-3-319-04663-1_4
14. Vukorep, I., Kotov, A.: Machine learning in architecture. In: *The Routledge Companion to Artificial Intelligence in Architecture*, pp. 93–109. Taylor & Francis, London (2021)
15. Vukorep, I., Zimmermann, G., Sablotny, T.: Robot-controlled fabrication of sprayed concrete elements as a cyber-physical-system. In: *Second RILEM International Conference on Concrete and Digital Fabrication*, RILEM Bookseries, pp. 967–977. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-49916-7_94